CS 470 Lab 2 Report

Noah Rodal

February 10th, 2025

Introduction

For this project, we were tasked with creating a C program that had us have a parent process create ten child processes, and have each of them use a simple execvp function. This lab is meant to demonstrate our knowledge of using commands such as fork(), wait(), and PID usage.

Implementation

For the implementation of this assignment, the C program was split into two functions: A main, and a execute function. The main function is where everything to create the child processes is done, while the execute function holds the commands for the child process to execute.

Starting with the main function, we first create an array of PIDs for the child processes using the pid_t type. Then a for loop is initiated going until the end of the array of PIDs creating child processes using fork(). Then an if statement checks to see if the PID of the child process is less than zero, and if so the program prints that the fork failed. If successful, the execute_command is function is called. In this function, a char array is made to hold the command ls, and then execvp is called using the char array to display the files in the current directory using the child process.

After this is done, the parent process is told to wait until the child process is finished, and the status of the child process is stored. If the child process finishes with a successfully, the program prints to the terminal that the process was successful, and gives its ID. If it is not successful, the program prints that it has been terminated.

Then the for loop would continue and the parent process would move on to the next iteration, and create another child process, until 10 of them were made

and used to execute the ls command.

Observations

Handling the processes was quite simple. A array of pid_t was used to have the parent process create, and then store the process id in to the array for further use. To catch the child finishing, WIFEXITED(status) was used to confirm whether the process was successful or not.

From a small amount of testing through using additional code, it was found that the parent process would execute its code first, if the wait() command was not used, and that everything after the wait command would be executed only after the child process was finished. With this knowledge, the wait() command always needs to be after the code the child uses to execute, as the parent will continue on until it hits a wait command.

Conclusion

This lab allowed us students to explore the use of fork(), wait(), and process management so that we can see how parent and child process interact with one another. Overall, the code for the project looked intimidating at first, but the amount of code that was required was surprisingly simple, and it will interesting to see how these commands are potentially used in future assignments to interact with one another in a way to create more complex programs.